User Manual
ImpacctLive v1.0

Dexin Li and Jiwon Hahn
Dept. of Electrical Engineering & Computer Science
University of California
Irvine, CA 92697-2625 USA
{dexinl,jhahn}@uci.edu

October 25, 2004

Contents

Introduction **2**

User Manual **2**

1 Overview 3

2 Input Format 6

2.1 Static Input 6

2.2 Dynamic Input 15

3 Dynamic Scheduler 17

3.1 Starting the CORBA naming service 17

3.2 Starting the dispatcher (CORBA server) 17

3.3 Running the scheduler 19

4 IMPACCT Simulator 21

4.1 Introduction to Graphical User Interface 21

4.2 Using the GUI 22

4.3 Running scheduler with simulator 25

A Software Installation **27**

B Glossary **29**

Introduction

ImpacctLive is a novel system-level modeling and simulation environment. It provides users with means that support all stages of designing a complex embedded system, from system specification to simulation and profiling. A key feature of *ImpacctLive* is its power optimization capabilities in terms of both static and dynamic scheduling and power mode selection for real-time embedded applications.

ImpacctLive provides a graphical user interface (GUI) that supports features including design authoring, mode simulation, system profiling, and report generation. The GUI helps users go over the entire design process with ease and visualize dynamic behavior of the system very intuitively. Moreover, users can easily modify system models and parameters and repeat the design cycle to explore the design spaces in a short while. Without the tool, the design space exploration is often tedious and time-consuming, if not impossible. The report automatically generated by the tools gives detailed power breakdown and power savings of subsystems, with which users are able to perform power analysis, identify design hot spots, and make meaningful decisions to improve power efficiency of subsystems.

In *ImpacctLive* version 1.0, a dynamic scheduler and a simulator are integrated together with CORBA client/server modules. At runtime, the scheduler generates power commands at runtime based on the prediction of system behavior in the near future. The simulator can work as a virtual hardware platform and execute power commands obtained through the CORBA interface from the scheduler. The CORBA IDL and the naming service are essential to the communication between the scheduler and the simulator, allowing the tool to work seamless on either a local host or distributed servers.

This user manual presents a brief overview and step-by-step tutorial for using the tool. It does not intend to give a comprehensive explanation of the underlying techniques and methodology. For such information, please contact the IMPACCT group. The manual starts with an overview of the tool in Section 1. The following sections go over the input format (Section 2), the scheduler (Section 3), and the simulator (Section 4). Guide for software installation and glossary are provided at Appendix A and B, respectively.

User Manual

1 Overview

ImpacctLive version 1.0 is an integrated environment for power optimization and simulation. It supports the following functions:

- System specification and design authoring
- Dyanmic scheduling and power optimization
- Communication between hardware and software through CORBA services
- Simulation environment with graphical user interface
- Report generation for power analysis

Figure 1 shows the entire tool set of *ImpacctLive*. For system specification, *ImpacctLive* provides an integrated GUI called *System Description Wizard* (SDW) that includes a resource editor, a resource instantiator and a task graph editor. The SDW can be used to specify the power attributes of resources in the system. As the resources get upgraded and enhanced with more power management features, they can be added to the power model. Additional resources can be added to the system architecture interactively, and the flow graphs for mapping the functional semantics to the new architecture can be graphically rendered by an automatic layout engine.

IMPACCT scheduler in Figure 1 can be seen as a framework that includes admission controller, timer, core scheduler, dispatcher and CORBA client. The framework is composed in a modular form so that power optimization policies implemented in different software modules can be flexibly plugged into the dynamic scheduler without affecting the rest of the tool. It is ideal to accommodate user defined policies to be evaluated in the same environment.

The simulation tool receives power commands generated by the scheduler and simulate the dynamic behavior of the entire system. The activities in every subsystems are profiled.

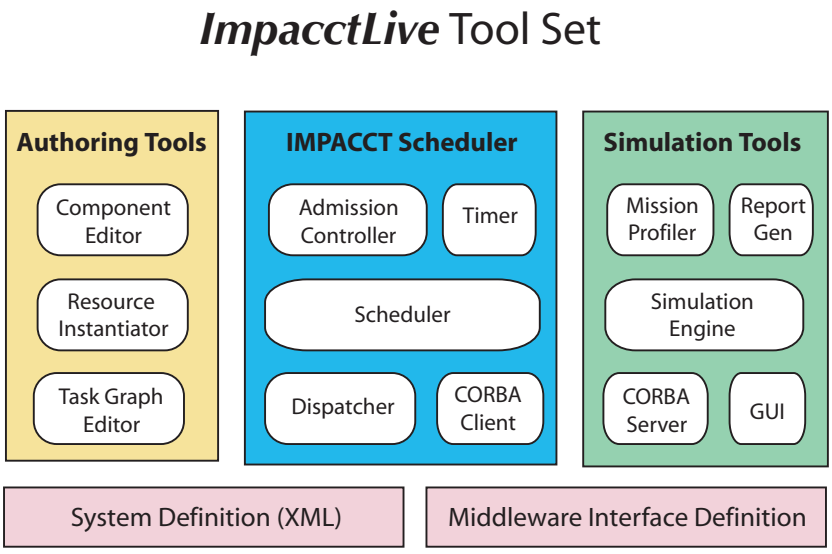

Figure 1: Composition of *ImpacctLive*

The front-end GUI of the simulator provides direct control over the simulation as well as CORBA commands to the actual JTRS radio. On the GUI, the power levels are color-coded and highlighted at the subsystem component level. In addition, a scenario-oriented animation can be viewed to observe the activities among subsystems. The results from the simulation are collected and displayed as either consolidated or detailed, variable-resolution power profiles that can be inspected for hotspots. Finally, a report-generator module integrated with the GUI automatically outputs power statistics from each experiment, and the views and breakdown options are customizable.

The data exchanged between the authoring tool and the scheduler are defined in the System Description Format (SDF) in XML. Once the system specification is updated, the system description in an XML file will be automatically generated and be ready for the scheduler to use as the system configuration.

The CORBA IDL defines the format of power commands transfering from the scheduler to the simulator. A CORBA naming service is required to be active for both sides to communicate with each other.

For the realtime operation, the scheduler takes the system configuration and the mission configuration defined in the SDF file as a static input and incoming messages as a dynamic input. The IMPACCT Scheduler schedules the messages according to the built-in power optimization policies, and generates power control commands for each subsystem. There are two ways to execute the power commands. If hardware subsystems defined with the

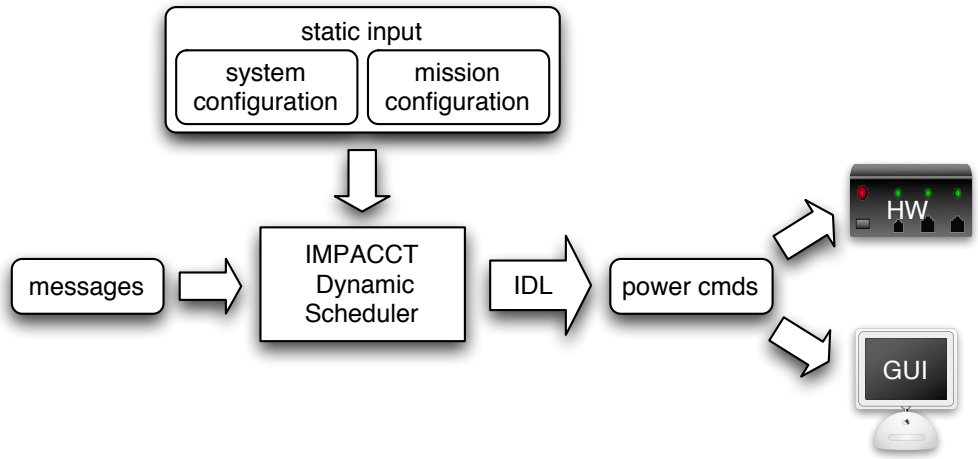

Figure 2: Data Flow of *ImpacctLive*

same CORBA IDL is in the loop, the power commands can be delivered to them through a dispatcher implemented as a CORBA client. In the other case, the simulator can be used to replace the real hardware and receive power commands from the dispatcher through the same CORBA IDL. Figure 2 illustrates the runtime configuration of the tool.

2 Input Format

The input to the *ImpacctLive* consists of two parts: static and dynamic. The static input corresponds to the architecture and application modeling, which are specifically system configuration and mission configuration, respectively. The dynamic input is the hints of incoming messages. We explain the format of these inputs in the following subsections.

2.1 Static Input

Before dynamically running the tool, users need to configure the target system and the mission parameters. Our system model is described in XML format, which can be either manually generated or rendered through a system definition wizard.

System Description

Our system model consists of resource library, architecture, and application. The resource library contains resource types, including operational and idle power modes and the mode transition power and timing overhead. The architecture contains the instances of resource types, which are the building blocks of the entire system. The architecture also specifies *taskgraphs*, which indicates existing path of a task through a sequence of resources. The application model contains different possible application scenarios, although this part is not currently in use. As an alternative for specifying the application model in the system model, we use the previous mission configuration format to represent a mission, and our mission parser to capture the corresponding scenario parameters.

An example of the system description is shown in Figure 3. More detailed and formal definition of this XML system model can be found at [2].

Authoring Tool: SDW

The system description wizard (SDW) is a user friendly tool for aiding the correct system construction that meets the format requirement. Through the step-by-step configuration, the user finally gets the XML system description explained in the above subsection as an output.

To start the tool, open your system shell, go to the directory you installed SDW, and execute `sdw.py`.

```
% cd SDW

% python sdw.py
```

```
<system xmlns="" name="JTRS_1Channel" fineClock="1">
    <resLibrary name="JTRS_Resources">
        <resType name="Transceiver">
            <idleModes>
                <im name="standby" pwr=".1" entTO="1000" entPO="25"
                    extTO="1" extPO="25" minDuration="1002"/>
                <im name="off" pwr=".01" entTO="1000" entPO="25"
                    extTO="10000" extPO="25" minDuration="11000"/>
            </idleModes>
            <DVSModes>
                <m name="on" pwr="25" frq="1"/>
            </DVSModes>
            <functions>
                <f name="p" duration="1" pFactor="1"/>
            </functions>
        </resType>
        ...
    </resLibrary>
    <architecture name="JTRS" profiling="false" centralPwrManager="true"
                  commandSlotSize="1">
        <res name="T1" resRef="Transceiver" profiling="false" DVS="false"
             ctrlByCPM="true">
            <port name="1"/>
        </res>
        ...
        <taskGraphs>
            <taskGraph name="1RR5">
                <root name="tPA1" res="PA1" inPort="1" func="5" />
                <node name="tRI" res="RI" inPort="1" func="p" />
                <node name="tRE" res="RE" func="p" />
                <node name="tR1" res="R1" func="p" />
                <node name="tE" res="E" func="p" />
                <node name="tBE" res="BE" func="p" />
                <node name="tB1" res="B1" func="p" />
                <node name="tM1" res="M1" func="p" />
                <node name="tT1" res="T1" func="p" />
                <edge dNode="tRI" sNode="tRE" />
                <edge dNode="tRE" sNode="tR1" />
                <edge dNode="tR1" sNode="tE" />
                <edge dNode="tE" sNode="tBE" />
                <edge dNode="tBE" sNode="tB1" />
                <edge dNode="tB1" sNode="tM1" />
                <edge dNode="tM1" sNode="tT1" />
                <edge dNode="tT1" sNode="tPA1" />
            </taskGraph>
            ...
        </taskGraphs>
    </architecture>
    <application name="">
        <phase>
            <task name="receiveRed" TG="4SB4" period="10000" start="0"
                  deadline="10000"/>
            ...
        </phase>
        ...
    </application>
</system>
```

Figure 3: Example of XML system description

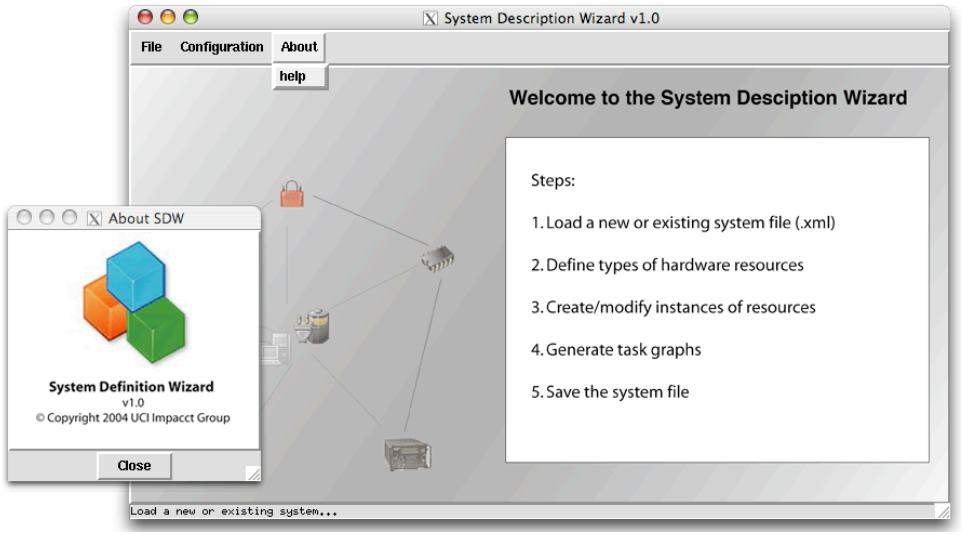

Figure 4: SDW - Start screen

Menu	Option	Meaning
File	New System	Open a new system (default model)
	Open...	Open an existing system
	Save	Save the current step
	Save As...	Save as different name
	Save All	Save modifications of all three steps
	Close	Close the current system
	Exit	Exit program
Configuration	Resource Type Definition	First step: define resource types
	Resource Instantiation	Second step: instantiate resource
	Task Graph	Third step: Generate & Manage the task graphs

Table 1: Menu options for SDW

A start screen will pop up, as shown in Figure 4. It displays the steps to follow for configuring your system. It has three configuration steps to follow The menu options are organized in Table 1.

First, the user should load either a new or existing system (Figure 5). The new system corresponds to a default model, which can be modified to create a new system. This tool is mainly targeted for modifying an existing system to either remove or add a component, although building a brand new system is still possible. After opening the system, the user can check the loaded file on the status bar on the bottom left, or by the title of the window.

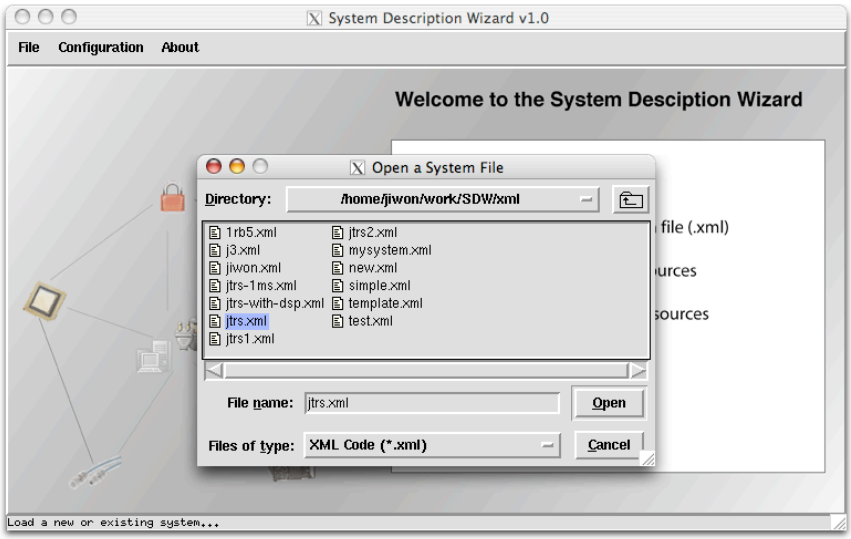

Figure 5: SDW - Open an existing system model

Next, the user enters the first configuration step by choosing ‘Resource Type Definition’ from the menu. In this step (Figure 6), the user can add/edit/remove a resource type on the leftmost frame. When a resource type is selected by clicking on the checkbox, the middle frame shows the corresponding power modes and the rightmost frame shows the mode transition overhead. Figure 6 shows an example of selecting a *Modem* and adding a *doze* mode, which is an *idle* mode with 0.5 W power consumption. The user can freely add/edit/remove any modes or transition overhead entries.

After defining the resource types, the user can create resource instances to build a system in the second configuration step. Figure 7 will be shown when choosing ‘Resource Instantiation’ in the Configuration menu. On the screen, the added instances are listed. Each line contains instance ID, resource type, and optionally channel and the power source of the instance. The user can add/edit/remove each instance.

When a group of instance are collected, the user may define task graphs. By choosing

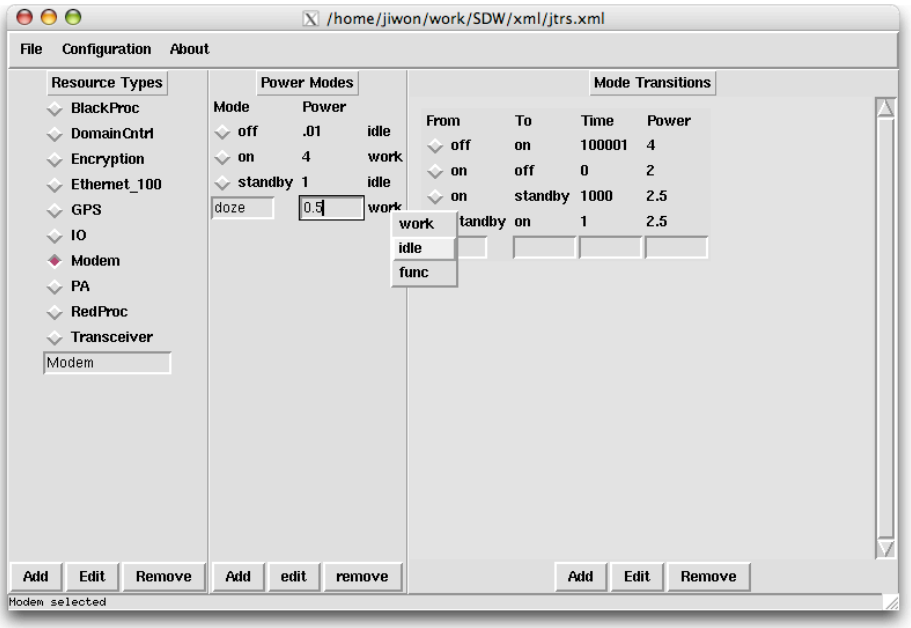

Figure 6: SDW - Resource Type Definition

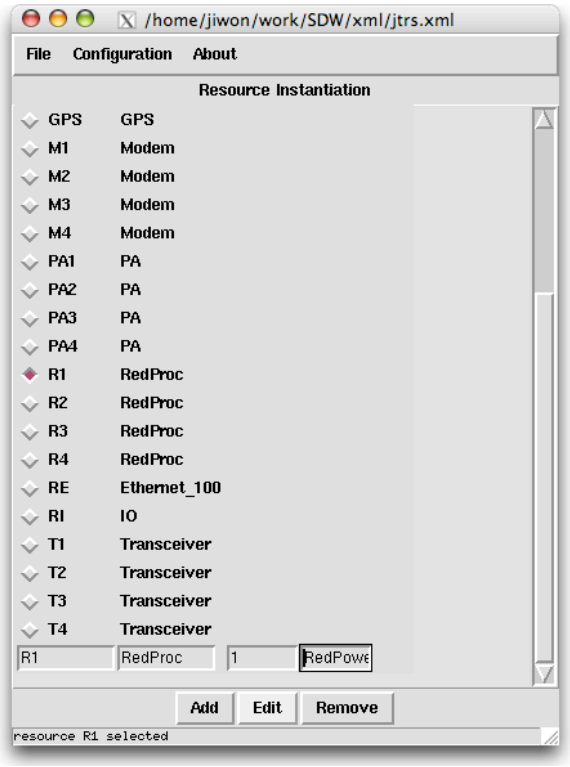

Figure 7: SDW - Resource Instantiation

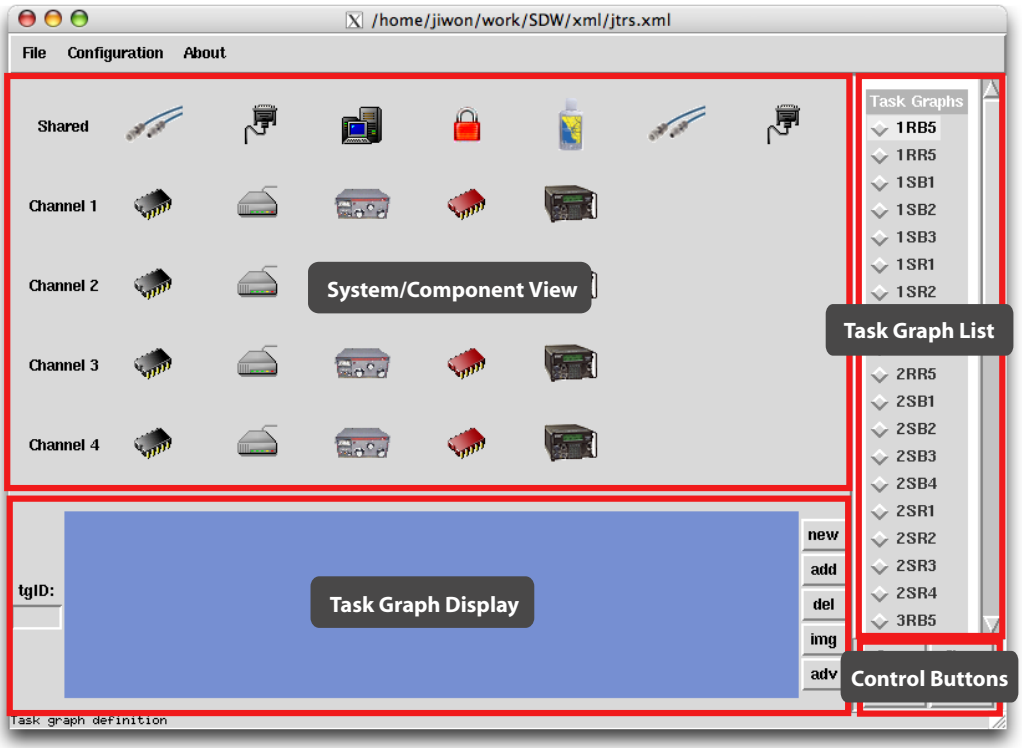

Figure 8: SDW - Task Graph Manager

'Task Graph' in the configuration menu, the user enters a screen as shown in Figure 8. The window consists of four frames, including system/component view, task graph display, task graph list, and control buttons.

The top right frame shows the entire system components arranged by channel. The ones that don't belong to any channel becomes shared resources. By rolling the mouse over the components, the resource type and instance ID pops up.

The bottom left frame displays the task graph details. Table 2 explains the usages of each button. The original blue canvas is for sequential task graphs only, thus called *basic* view. In this screen, user can generate task graphs by clicking on the components on the system/component frame (Figure 9). Users can undo the current component selection by double-clicking on the blue screen. When the task graph is completely drawn, the user may enter a tgID (task graph ID) on the left and add it to the list by clicking the 'add' button.

Another way of generating a task graph is by modifying an existing task graph. The top right frame shows the current list of task graphs. When the user selects a tgID from the task graph list, the task graph is visualized on the displayed screen. Figure 10 shows an example of selecting a task graph called '1SR1'.

The task graph may not be sequential, but also in a tree structure (eg., a resource con-

Button	Usage
new	Clear the task graph display screen
add	Add the task graph on the display to the task graph list
del	Delete the task graph on the display from the task graph list
img	Render the task graph with a graph drawing engine
adv	Switch to advanced view
bas	Switch to basic view

Table 2: Task Graph Display Control

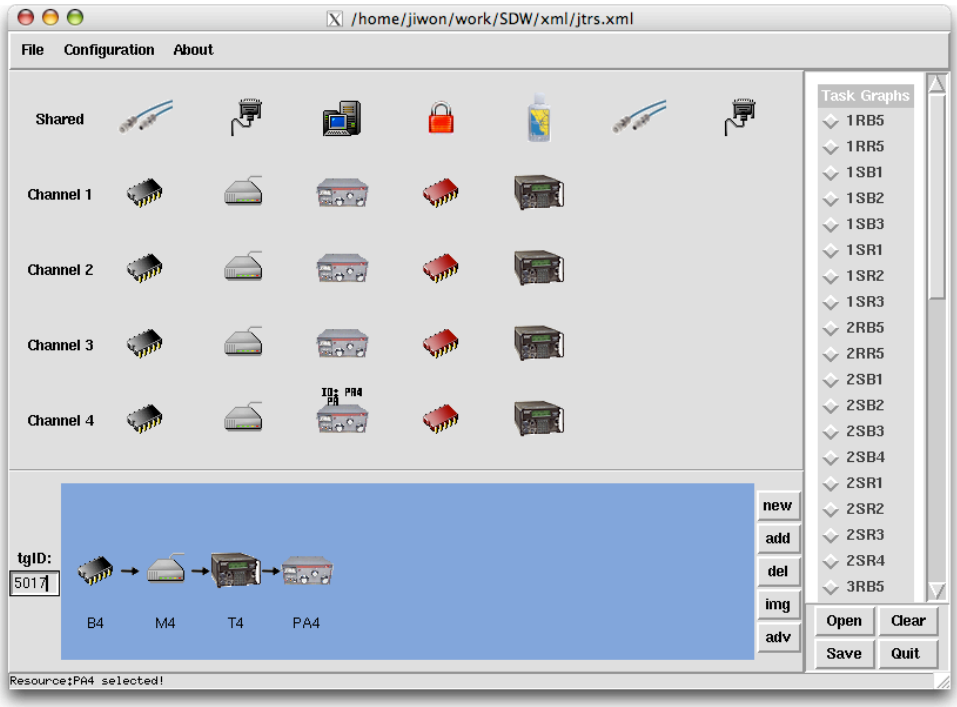

Figure 9: SDW: Simple Task Graph Generation by Component Selection

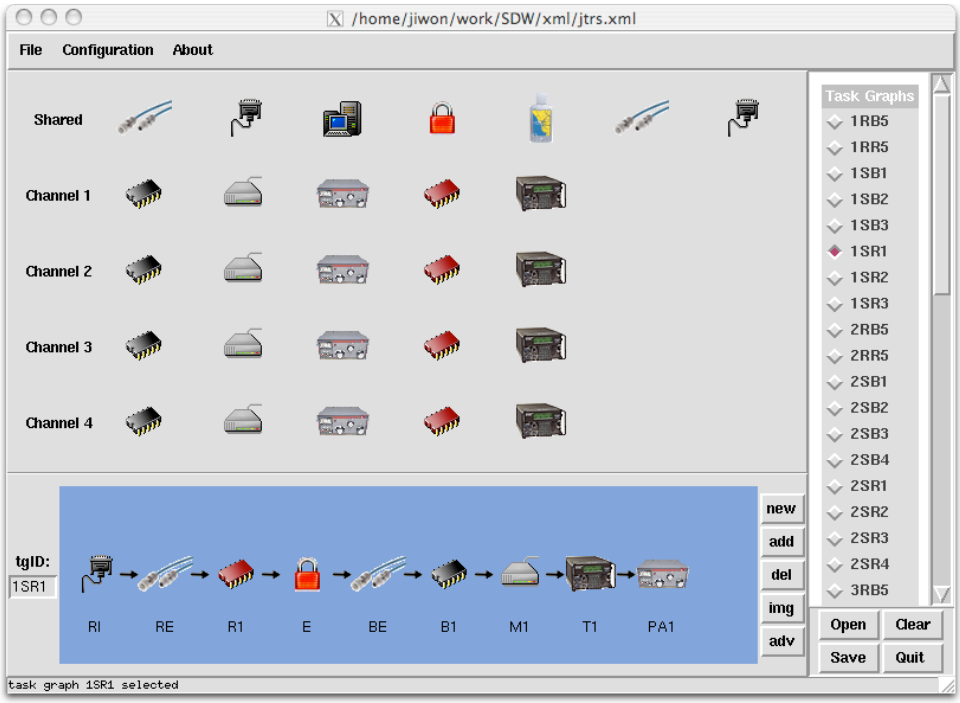

Figure 10: SDW: Task Graph Display by tgID selection from the list

Figure 11: SDW: Alternative Task Graph Display (*advanced* view)

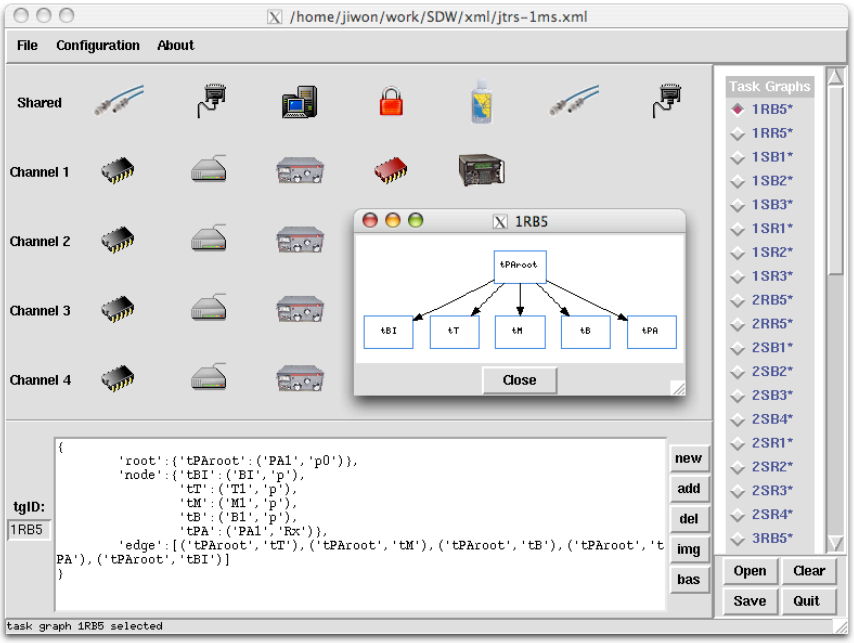

Figure 12: SDW: Task Graphs with Tree Structure

nected to more than one resource). For a task graph with a tree structure, the user needs to switch to *advance* view by clicking on 'adv' button (Figure 11). The task graphs that are already in a tree structure shows in blue text attached with '*' (Figure 12). When clicking on such tgID, an advanced view with the code in python shows. It contains a very simple dictionary structure, where user can directly make modifications to the complex task graph. By clicking on the 'img' button, the user can check the pictorial view of the tree from the pop-up window (Figure 12). It is generated by a graph drawing algorithm called DOT [1].

The control buttons on the bottom right let the user open an existing task graph, clear the current task graph list, save the current list, and quit the program.

Before terminating the program, the user should save all the configurations by clicking 'Save All' in the File menu, and the user will have the `system_name.sys` ready to plug into the scheduler.

Mission Configuration

The mission configuration is required for extracting the mission information, which will provide additional power saving opportunity. For the current implementation, we only extract the opcode and type mapping information and the format is shown in Figure 13.

The mission configuration file (`mission_name.xml`), which was used for the static scheduler [3] can also be used without any modification.

```
<messages>
   <message opcode="401" type="request"/>
   <message opcode="501" type="response"/>
   <message opcode="403" type="request"/>
   <message opcode="503" type="response"/>
   <message opcode="410" type="request"/>
   <message opcode="510" type="response"/>
   <message opcode="411" type="request"/>
   <message opcode="511" type="response"/>
   <message opcode="412" type="request"/>
</messages>
```

Figure 13: Example of Mission Configuration (`mission.xml`)

2.2 Dynamic Input

We have defined an API between the mission computer and *ImpacctLive*. Mission computer is an external entity that sends the mission messages (commands) to our *ImpacctLive*. The API defines the format of the mission commands from the mission computer, and the acknowledgement back to the mission computer. Table 3 shows the mission command format, and Figure 14 is an example file. The file format is `filename.cmd`.

Field	Meaning
time	timestamp when the real message will arrive
message ID	unique key for the message. Currently, it's specified as *number-number*, where the first number indicates the opcode ID, and the second number shows the count for that opcode
opcode	specific operation the message encodes
channel	hardware path the message is mapped to
period	the arrival period of the specific opcode
distance	communication distance at the timestamp
from	source of the message (eg., 0:Mision computer)
to	destination of the message (eg., 1:*ImpacctLive*)
cmd	types of command (eg.,1:INSERT 2:DELETE, ...)
altch	the alternative channel assignment in case of failure
seq	tagged sequence number for the outgoing messages
ref	reference ID for acknowledgement
app	N/A

Table 3: Message format from mission computer to *ImpacctLive*

For now, there is no established model for the mission computer, so we tested with the same mission profiles from the static scheduler [3]. This file is the output of the MAPMgen

time	msgID	opcode	ch	period	dist	from	to	cmd	altch	seq	ref	app
18.1	51-13	57f	1	1.5	200.60	0	1	1	-1	-1	-1	1
18.1	57-4	5fe	1	3.0	200.60	0	1	1	-1	-1	-1	1
19.5	22-14	47f	1	0.5	200.64	0	1	1	-1	-1	-1	1
19.6	51-14	57f	1	1.5	200.65	0	1	1	-1	-1	-1	1
20.0	59-1	498	2	3.0	400.40	0	1	1	-1	-1	-1	1
20.0	1-3	202	3	31.5	600.49	0	1	1	-1	-1	-1	1

Figure 14: Example message commands from mission computer to *ImpacctLive* (`mission.cmd`)

and must be saved as `filename.mapm`. The user should input the `.mapm` file, and our mapm2cmd converter will automatically generate `filename.cmd`, which will be dynamically dispatched to *ImpacctLive*.

3 Dynamic Scheduler

The IMPACCT Dynamic Scheduler (scheduler for short) schedules the power modes of resources based on prediction of future messages within a bounded windows. The input to the scheduler is messages within the sliding window of the near future along the time-line. The output of the scheduler is a set of power commands for each resource with time stamps. Before the commands are committed, it is possible for the scheduler to re-schedule the commands based on the dynamic changes of messages at the runtime. Once the commands are committed, they cannot be re-scheduled and are ready to be dispatched through a CORBA interface at real-time to a CORBA server that has direct control over the hardware devices.

The scheduler can be executed in two modes: with or without the IMPACCT Simulator (the simulator for short). The simulator works as a virtual hardware platform. It receives power commands from the scheduler, simulates the runtime behavior of the system, and collects detailed power statistics of the system. Without the simulator, a simple CORBA server can be set up to receive power commands from the scheduler. In either case, CORBA naming service is needed for both the client side (the scheduler) and the server side (the simulator or the simple server) to communicate with each other.

This section describes the steps to run the scheduler without the simulator. The steps to run the scheduler with the simulator is to be presented in the next section, after the introduction to the simulation and the graphical user interface (GUI).

3.1 Starting the CORBA naming service

CORBA naming service must be present before starting the simulator or the scheduler. The host and port setting of the CORBA name service can be specified in file "ns_remote_host.cfg" and "ns_remote_port.cfg" in the "$JTRSHOME/src/dynamic_scheduler/dispatcher" directory. The default value for "ns_remote_host.cfg" and "ns_remote_port.cfg" are `localhost` and `10000`, respectively. To start the naming service, run the following command in a terminal:

```
% $JTRSHOME/src/dynamic_scheduler/dispatcher/start_ns.sh
```

Figure 15 shows the output text when the naming service is started successfully.

3.2 Starting the dispatcher (CORBA server)

In a new terminal, start the dispatcher by running the following command:

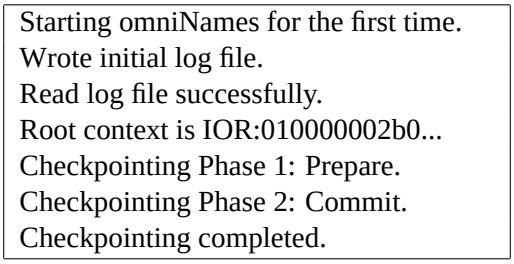
Starting omniNames for the first time.
Wrote initial log file.
Read log file successfully.
Root context is IOR:010000002b0...
Checkpointing Phase 1: Prepare.
Checkpointing Phase 2: Commit.
Checkpointing completed.

Figure 15: Output message if CORBA naming service is started successfully.

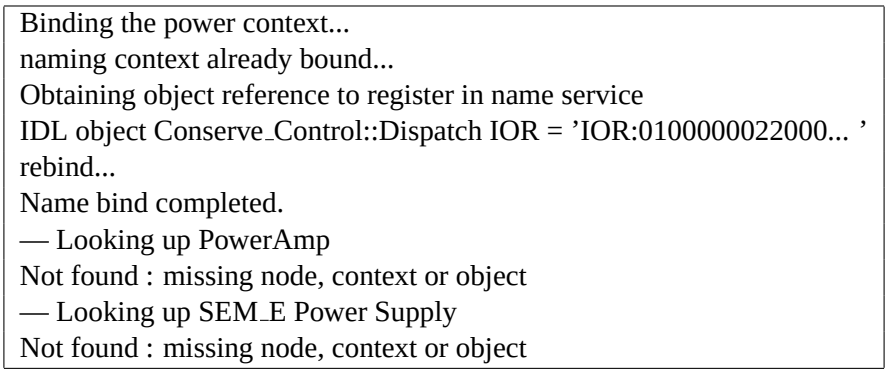
Binding the power context...
naming context already bound...
Obtaining object reference to register in name service
IDL object Conserve_Control::Dispatch IOR = 'IOR:0100000022000... '
rebind...
Name bind completed.
— Looking up PowerAmp
Not found : missing node, context or object
— Looking up SEM_E Power Supply
Not found : missing node, context or object

Figure 16: Output message if the CORBA server is started successfully.

```
% cd $JTRSHOME/src/dynamic_scheduler/dispatcher

% ./start_server.sh
```

The dispatcher will open a CORBA session and wait for the connection from the client (the scheduler); however, it will only receive power commands without simulating the system and calculating the energy savings. Figure 16 shows the output text when the CORBA server is started successfully.

If either the scheduler or the dispatcher fails, please check whether the environment variables are set up correctly.

Once the dispatcher receives power commands from the scheduler, it displays all the commands on the terminal screen. Users may want to save the power commands by executing:

```
% ./start_server.sh > [result_file]
```

to redirect the screen output to a file.

3.3 Running the scheduler

Starting the scheduler

Open another terminal under directory "src/dynamic_scheduler" and type command:

```
% ./dynsch [configFile]
```

to start the scheduler.

Configuration file format

[configFile] is a configuration file that specifies the mission profile and the scheduler's settings. Table 4 shows an example of the configuration file "sim-5min.cfg" in the directory "src/dynamic_scheduler". This file contains configuration information about a five-minute example mission to be used as an input by the dynamic scheduler at the runtime. Another configuration file "sim-10hr.cfg" in the same directory is provided for testing the ten-hour mission.

In these files, line 1 specifies the mission profile. The mission profiles are under "src/dynamic_scheduler/mission" directory. Line 2 specifies the JTRS system architecture and power models stored in an XML format file. The default location is the directory "src/dynamic_scheduler/scheduler". Line 3 provides the CORBA command line options used by the CORBA client to be connected to the dispatcher. The default host name is `localhost` and the default port number is `10000`.

The speedup factor (line 4) should be 1 to allow the mission to be simulated in real time. Users may wish to apply a larger speedup factor (e.g., 10) that allows to finish a long mission in 10% of the actual time. However, the program may not complete correctly if the speedup factor is too large (> 50).

Users can modify the (unit) window size (line 5) to change the size of time intervals that the scheduler works with. That is, the scheduler can schedule all messages that arrive in an interval whose length is equal to the window size.

Committed window size, scheduled window size and updated window size (line 6–8) are specified as multiples of the unit window size. For example, if the window size is 1 second, and the committed window size is 2, the committed window is $1 \times 2 = 2$ seconds. If the window size is reduced to 0.5 second, the committed window is $0.5 \times 2 = 1$ second. The current settings for committed window size, scheduled window size and updated window size should work fine with various values of unit window sizes.

Line 9 specifies whether to use boundary pattern matcher. Setting this variable helps reduce the total number of power commands generated by the scheduler.

line #	Configuration value	Configuration meaning
line 1	mission/5min.cmd	mission profile
line 2	scheduler/jtrs.sys	JTRS system xml
line 3	-ORBInitRef NameService=corbaname::localhost:10000	CORBA command line options
line 4	1	speed up factor
line 5	1	window size
line 6	2	committed window size
line 7	2	scheduled window size
line 8	5	updated window size
line 9	1	whether to use boundary pattern matcher

Table 4: An example configuration file (`sim-5min.cfg`).

Checking the execution

When the scheduler is running, some messages with numbers are displayed on the screen, indicating the execution of the scheduling thread and the dispatching thread. Meanwhile, on the dispatcher side (the CORBA server), the received power commands are displayed on the screen. If the user redirects the output to a file, the power commands will be saved in the file without printing on the screen.

Stopping the dispatcher

After the scheduler finishes its execution, the dispatcher must be terminated by hitting "Ctrl+C" to terminate the CORBA server session.

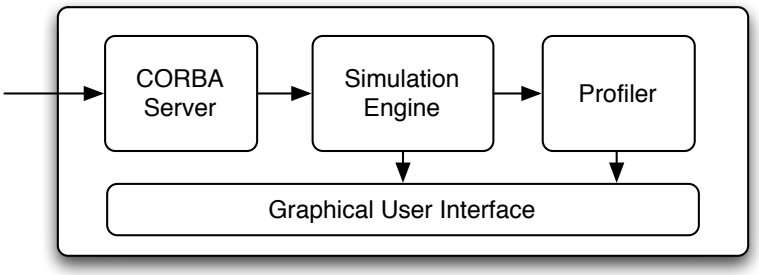

Figure 17: The IMPACCT Simulator.

4 IMPACCT Simulator

The IMPACCT Simulator (simulator for short) is a software module that simulates, visualizes and reports behavior of the scheduled mission on the system model. The simulator can work as a virtual hardware platform. It simulates the system behavior and generates power statistics of each resource. The system status is updated at runtime, which can be visualized in a graphical user interface. The user can also load a pre-stored profile to playback a simulated mission.

The simulator consists of a CORBA server, a simulation engine, a profiler and a graphical user interface (see Figure 17). The CORBA server runs at the backend to receive power commands from the scheduler, using the same IDL as the dispatcher does. The simulation engine executes the power commands and sends the system status information to both the graphical user interface and the profiler at runtime. The profiler also runs at the backend and collects a variety of statistical information.

In the rest of this section, we will introduce the graphical user interface in more depth, and describe how to use the simulator together with the scheduler.

4.1 Introduction to Graphical User Interface

Main Window

A snapshot of the Main Window during simulation is shown in Figure 18. At the top of the window are the Main Menu and the Toolbar. The Toolbar provides shortcuts for some of the simulation-related functions, as shown in the figure. The Profile Window right below the toolbar shows the power profile for the simulated mission. The Resource Window on the bottom left shows the resources in the system. In this figure, each row represents a channel in the JTRS system except the last row for the shared resources. Different colors encode

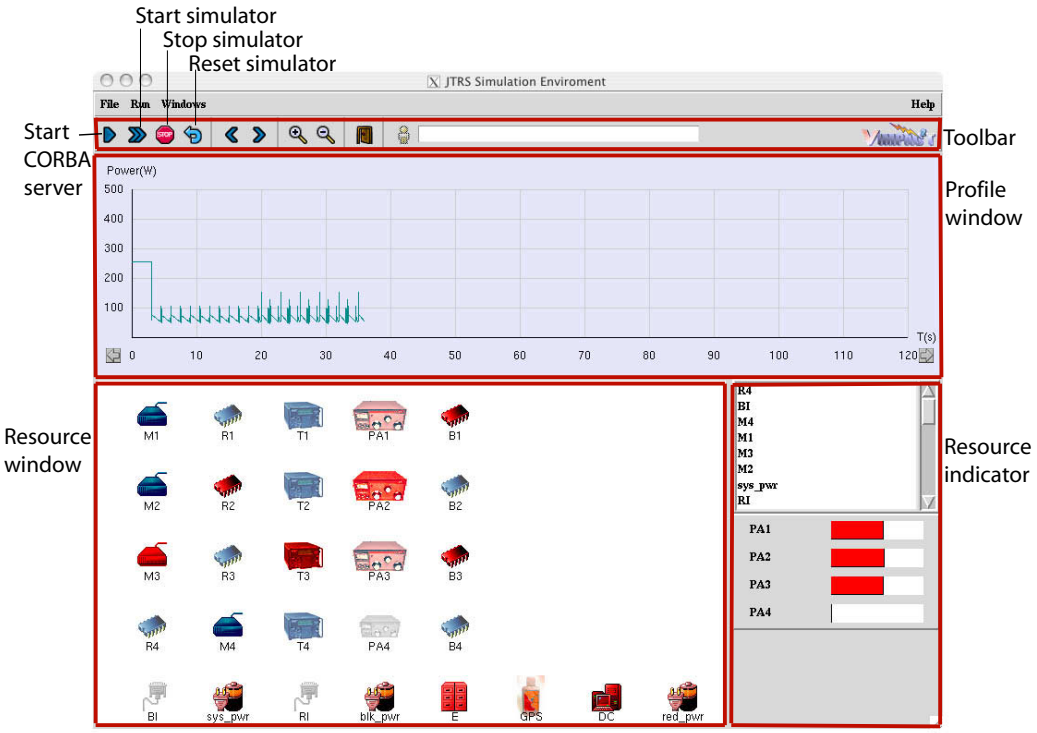

Figure 18: The main window of the simulator GUI.

power modes of resources, red being the highest power mode. The Resource Indicator on the bottom right shows the power levels of selected resources.

Main Menu

The functions of the Main Menu are summarized in Table 5. One thing to note is that the user can select Window→Control Panel to turn on the Control Panel, with which the user can have quick accesses to some of the menu functions.

Control Panel

The Control Panel is shown in Figure 19. The mapping between the icons and the corrensponding menu functions are explained in Table 6.

4.2 Using the GUI

The GUI of the simulator is able to play back a simulated mission profile, provide views for power analysis and report generation. Most importantly, it supports dynamic simulation,

File Menu	Description
File→Load Library	Load the component library, ie., `mycomplib.py`
File→Load Profile	Load a pre-simulated mission
File→Save Profile	Save the current mission profile
File→Exit	Exit the Program
Run→Dynamic Simulation	Start the CORBA server, ready for simulation
Run→Step Forward	Step forward in a simulated profile
Run→Step backward	Step backward in a simulated profile
Run→Backward to Begin	Reset the simulation profile to the starting point
Run→Stop Simulation	Stop simulation
Run→Generate Report	Generate simulation report
Window→Resource Power Profile	Display the power profiles of each resource
Window→Energy breakdown	Display pie charts of energy breakdown
Window→Control Panel	Toggle the display of the Control Panel

Table 5: The Main Menu functions.

Figure 19: The Control Panel.

where the profile is continuously updated on the screen, reflecting the current results of the back-end simulator.

Simulation playback

To play back a simulated mission profile, select from the Control Panel or select `File→Load Profile` from the Main Menu to load a mission profile. Press in the Toolbar to start playback the mission profile. Press in the Toolbar to stop the playback.

Alternatively, when a mission profile is loaded, the user can perform step simulation by Step Forward or Step Backward from the Toolbar. At any moment, the user can change the scale of the Profile Window by Zoom In or Zoom Out .

Icon	File Menu
	File→Load Library
	File→Load Profile
	Run→Dynamic Simulation
	Window→Resource Power Profile
	Window→Energy breakdown
	Run→Generate Report
	File→Exit

Table 6: The function mappings of the Control Panel.

Viewing power profile

To display the power profiles of channels and resources, Select in the Control Panel or `Window`→`Resource Power Profile` from the Main Menu. A new window of channel-level power profiles will show up, where each track represents the power profile of a channel. The user can zoom in or zoom out the power profile by clicking the corresponding buttons at the top of the window. By double-clicking a track, the user can view a finer grain (component-level) power profiles.

	View more details of the profiles
	View less details of the profiles
	Exit the window

Viewing pie chart

To display the pie charts, select `Window`→`Energy Breakdown` in the Control Panel. The pie chart shows the power break-down of the system by channels. Double-click each portion in the pie chart to view each channel's energy break-down by components.

Report Generation

The user can use the report generation tool to generate customized text reports and figures. Start the report generation tool by selecting `Tools`→`Generate Report` or by clicking in the Control Panel. The user can generate the following items in the text report:

- General mission information
- Channel-level energy information

- Component-level energy information
- Channel-level command information
- Component-level command information

The user can set the sorting criteria for the energy information by channel, by component name, by energy usage, or by time usage. The user can also set the sorting criteria for the command, information by channel, by component name, or by command count. The user can save the text report by clicking the `Save` button. To load a previously stored text report, click Load button. Table 7 is the description of the fields in the report.

4.3 Running scheduler with simulator

The scheduler and the simulator can be used together to simulate the dynamic behavior of the system. In this case, the simulator works as a virtual hardware platform, accepting power commands from the dispatcher and executing the command dynamically.

To start dynamic simulation with the GUI window, enter directory "src/gui/code" and type command:

```
% python jtrsgui.py
```

Now we need to start the CORBA naming service, if it is inactive. Pease refer to Section 3.1 for more information about how to start the naming service. In the Main Menu select `File`→`Load Library` to load a library file. The default library file is "src/gui/complib/mycomplib.py". Start the built-in CORBA server by clicking in the Control Panel or by selecting `Run`→`Dynamic Simulation` from the Main Menu. In a separate terminal, start the scheduler under directory "src/dynamic_scheduler" by type command:

```
% dynsch [configufile]
```

where [configfile] is a configuration file whose format was explained in Section 3.3.

The simulation should be starting at this point. If everything works successfully, the power commands are sent to the simulator and the Resources Window of the GUI is refreshing the power modes of resources. When the scheduler finishes sending commands to the simulator, click the Stop button in the Toolbar to terminate the dynamic simulation.

The simulation result can be saved by selecting `File`→`Save Profile` from the Main Menu. The saved profile can be used for playback later.

Field	Description
General	
Mission Name	The name of the mission
Mission Length	The length of the mission in seconds
Total No. of messages	Number of messages in the input mission profile
Total No. of commands	Number of power commands generated by the tool
Total waveforms	total number of waveforms
Total channels	total number of channels
Total components	total number of power-manageable components
System peak power	Peak power consumption over the mission, in Watt
Total energy	Total energy after power management, in Joule
Baseline Energy	Energy consumption without power management, in Joule
Energy savings	Energy savings in percentage
Channel-level energy statistics	
Channel	Channel number (0 is shared resources)
Energy	Energy consumption of the channle, in Joule
Percentage	Energy percentage of the channel
Component-level energy statistics	
Component Name	Component description
Energy(J)	Energy consumption of the component, in Joule
Energy(%)	Energy percentage of the component, in percentage
Ontime(s)	Time in which component has non-zero workload, in second
Ontime(%)	Time percentage of utilization time, in percentage
Channel-level command statistics	
Channel	Channel number (0 is shared resources)
Command(#)	Number of commands for that channel
Command(%)	Percentage of the number of commands
Component-level command statistics	
Component Name	Component description
Command(#)	Number of commands for that component
Command(%)	Percentage of the number of commands

Table 7: Fields Descrption for the Report

Appendix A

Software Installation

The current version of IMPACCT software is tested with the following setups:

- AMD Athlon XP 2400+ 1.8GHz, Fedora Linux with GCC 3.3
- Intel Pentium-4 Prescott 3.2GHz HT, Fedora Linux with GCC 3.3
- Intel Pentium-3 500MHz, Redhat Linux with GCC 3.1

Before building the IMPACCT scheduler and simulator, the following software packages must be installed to the target Linux system.

- OmniORB 4.0.3
- Python 2.3.3 or higher version
- Xerces-C 2.5.0
- Pathan 1.2

Obtaining IMPACCT Software Package

Here is the quick download link for the IMPACCT Scheduler package:

```
http://www.ece.uci.edu/impacct/software/JTRSAug.tgz
```

Unpack the entire suite of software packages and source code by command:

```
% tar xzf JTRSAug.tgz
```

under a desired directory and set this directory as the environment variable JTRSAUG-BASE, which is defined in file “JTRSAug/setup.csh”:

```
setenv JTRSAUGBASE $HOME
```

By default, JTRSAUGBASE is the user's home directory. This assumes that the software suite is unpacked under the user's home directory. Users should make changes accordingly if the entire software suite is unpacked under an alternative directory.

Installing the required software packages

All software packages are under "packages" directory. Refer to file "README.txt" under directory "packages" for detailed procedure to install all packages. The file "JTRSAug/setup.csh" may need to be changed, if users wish to unpacked/installed any packages to directories other than the default directory "packages". After all packages are installed, it is convenient to add the following line:

```
source [path_to_JTRSAug]/JTRSAug/setup.csh
```

into file "cshrc" under the user's home directory, then open a new terminal to let the new environment variables take effect. Otherwise, the same command must be executed every time before building and running the IMPACCT software.

The setup procedure is tested under C shell "tcsh". Users may need to change the command format of the setup script if a different shell, e.g., B shell "bash" is preferred.

Building the IMPACCT dynamic scheduler and simulator

The IMPACCT dynamic scheduler and simulator can be built under "src" directory by executing command:

```
% make
```

The executable file of the IMPACCT dynamic scheduler is "dynsch" under directory "src/dynamic_scheduler". The executable file of the IMPACCT simulator is "sim" under directory "src/simulator".

If errors are encountered during the build process, please check whether the environment variables are set up correctly by executing

```
% source [path_to_JTRSAug]/JTRSAug/setup.csh
```

or including this command in the "cshrc" file.

Appendix B

Glossary

CORBA	Common Object Request Broker Architecture
IOR	Interoperatable Object Reference
IMPACCT	Integrated Management of Power-Aware Computation and Communication Technology
IMPACCT dispatcher	The CORBA server that receives the power commands from the IMPACCT scheduler
IMPACCT scheduler	The power-aware scheduler developed by UC Irvine's IMPACCT group
IMPACCT simulator	A simulation engine integrated with a CORBA server and a graphical user interface to simulate and display the system behavior
MAPMgen	Mission profile generator developed by Rockwell Collins
Mision Computer	to be added
Mission Configuration	A mission scenario specified by start time, phases, waveforms, message indices, etc, captured in a pre-defined XML format
Mission Profile	A trace of messages ordered by timestamp, which specifies the released time of a message.
Task graph	to be added
Window, committed	to be added
Window, scheduled	to be added
Window, updated	to be added

Bibliography

[1] AT&T Labs-Research. Graphviz. In *http://www.research.att.com/sw/tools/graphviz/*.

[2] B. Gorji-Ara. System definition format. In *http://www.ece.uci.edu/∼bgorjiar/projects/jtrs/SDF-Doc/sdf.html*.

[3] J. Hahn, D. Li, and Q. Xie. User manual: Vimpacct v2.0. Technical Report 15-12-03, University of Calfornia, Irvine, December 2003.